Embed More Ignore Less (EMIL): Exploiting Enriched Representations for Arabic NLP

Ahmed Younes and Julie Weeds
Department of Informatics, University of Sussex
Brighton, BN1 9RH, United Kingdom
{ay227,juliewe}@sussex.ac.uk

Abstract

Our research focuses on the potential improvements of exploiting language specific characteristics in the form of embeddings by neural networks. More specifically, we investigate the capability of neural techniques and embeddings to represent language specific characteristics in two sequence labeling tasks: named entity recognition (NER) and part of speech (POS) tagging. In both tasks, our preprocessing is designed to use enriched Arabic representation by adding diacritics to undiacritized text. In POS tagging, we test the ability of a neural model to capture syntactic characteristics encoded within these diacritics by incorporating an embedding layer for diacritics alongside embedding layers for words and characters. In NER, our architecture incorporates diacritic and POS embeddings alongside word and character embeddings. Our experiments are conducted on 7 datasets (4 NER and 3 POS). We show that embedding the information that is encoded in automatically acquired Arabic diacritics improves the performance across all datasets on both tasks. Embedding the information in automatically assigned POS tags further improves performance on the NER task.

1 Introduction

Named Entity Recognition (NER) and Part-of-Speech (POS) tagging have traditionally been used as preprocessing steps in many Natural Language Processing (NLP) applications. For example, Yadav and Bethard (2018) discussed the use of NER across question answering, information retrieval, co-reference resolution, topic modeling, and machine translation. Similarly, POS tagging is often applied early in the NLP pipeline for many applications including information retrieval systems, syntax, and semantic analysis, speech recognition systems and machine translation (Abumalloh et al., 2016). In recent years, Arabic has been studied increasingly due to the explosion in the number of Arabic users on social media and the internet in general. Arabic is a morphologically rich language with complex grammatical structure (Shaalan et al., 2019). Arabic NLP researchers have used two types of approaches and sometimes a mixture of both to work with Arabic text. The first approach is the simplification approach where researchers tend to apply preprocessing (transformation) that simplify Arabic text such as letter normalization (Habash, 2010) and transliteration (Ameur et al., 2017). The second approach is the enrichment approach where researchers tend to apply minimum modification to the Arabic text and devise a way of incorporating the enriched features and potentially add more features to it.

We assume that the simplification approach may exclude some useful information, and hence take an enrichment approach. First, we add diacritics information inferred by automatic diacritization model called Shakkala, based on the assumption that the syntactic and semantic information encoded in diacritics might be useful in both the NER and the POS task. Once we have a diacritic-enhanced POS model, we use it to infer POS information for the NER corpora, based on the assumption that both diacritics and POS information can potentially improve the performance of the NER model. We are aware that this pipeline will raise the question of the quality of the inferred information and its effect on performance. Nevertheless, the experimental results shows that the addition of this automatically-inferred information enhances the performance.

Proceedings of the Fifth Arabic Natural Language Processing Workshop, pages 139–154
Barcelona, Spain (Online), December 12, 2020

Our specific contributions are that we propose a framework (EMIL) where we *Embed More and Ignore Less*. We show that applying minimal modification to the text and embedding more of the possible features into the model can outperform the standard sequence labeling models. We propose a diacritic-aware architecture for sequence labeling which extends and outperforms the current standard character-aware architecture of a Bi-directional long short-term memory network (Bi-LSTM) with a CRF. We also propose a combination architecture for NER that combines word, character, diacritic, and POS information and outperforms the standard character-aware architecture and our own diacritic-aware architecture.

2 Background and related work

We will now discuss the main characteristics of the Arabic language Section (2.1) and related work on Arabic NER and POS tagging Section (2.2).

2.1 The Arabic language

As discussed elsewhere, e.g., by Farghaly and Shaalan (2009), Arabic is rich morphology language with complex grammar structure which poses extra challenges to systems when considering Arabic text as input. Habash (2010) discussed the script differences such as letter shaping, script direction (right to left) and obligatory ligatures. Also Habash et al. (2013) discussed the lack of standard orthographies: e.g., غرام and جرام both mean (*gram*). One of the major challenges in Arabic NER is the lack of capitalization (Shaalan, 2014; Benajiba et al., 2008a). Shaalan (2014) also discussed the agglutinative nature of the Arabic language where new words and sometimes even sentences can be derived by adding affixes and clitics to Arabic words, making Arabic a morphologically rich language.

In our work, it is crucial to note that Arabic employs diacritics (short vowels) to encode phonetic, morphological, syntactic and semantic information. In traditional Arabic text, diacritics are symbols placed on top of Arabic letters. Figure (1) shows examples of three Arabic diacritic symbols. The choice of the diacritic on the last letter of the word describes the syntactic dependency of that word within the sentence. The choice of the diacritic on the first and the middle letters of the word disambiguates between different possible semantics of the word within the sentence. For example, the undiacritized word علم can be diacritized to become عَلِمَ (*knew*), عُلِمَ (*known*), or عَلَمٌ (*flag*). However, due to the fact that most modern Arabic text is only partially diacritized or undiacritized, researchers often remove it for consistency (Habash, 2010). Thus one word in Arabic may be ambiguous and the reader must use the context to disambiguate.

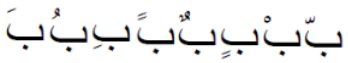

Figure 1: Arabic diacritics include: short vowels (first three letters), nunation (the second three letters), sukon (no vowel) and shadda (geminition) (the last two letters respectively).

2.2 Arabic Sequence Classification

Early work on the detection and classification of Named Entities in Arabic NER used a rule-based or grammar-based approach (Mesfar, 2007; Shaalan and Raza, 2007). Subsequently, the field shifted generally to a machine learning approach — thus avoiding the time-consuming and expensive maintenance of rule-sets. Within this general approach a wide variety of techniques have all been applied to the Arabic NER problem including Support Vector Machines (SVM) (Benajiba et al., 2008b), Conditional Random Fields (CRF) (Abdul-Hamid and Darwish, 2010; Benajiba et al., 2008a; AbdelRahman et al., 2010), Maximum Entropy (ME) (Benajiba et al., 2007), Hidden Markov Models (HMM) and Decision Trees (Nadeau and Sekine, 2007). Notably, Benajiba et al. (2007) developed an Arabic NER system, ANERsys 1.0, which employed maximum entropy and could recognize four types of Named Entities.

The authors also built their own linguistic resources, *ANERcorp* (an annotated corpus) and *ANERgazet* (a gazetteer), which have become benchmarks for evaluation. At this time, work was also done on incorporating POS information to improve NER. For example, Benajiba and Rosso (2007) proposed

ANERsys 2.0, where they used a POS tagger and a two step approach to enhance the performance of ANERsys 1.0.

More recently and following the general trend towards neural approaches, Gridach (2016) developed a character aware neural network model which attempts to capture contextual characteristics in Arabic by placing a CRF on top of a Bi-LSTM. This provided a hard state-of-the-art for other systems to beat and provides the foundation of our own approach. Very recently, Ali et al. (2019) applied a neural network model with a multi-attention layer to extract Arabic NEs. They used two attention units, the embedding attention layer, and the self-attention unit. They achieved an F1 score of 91.31 to achieve a new state-of-the-art on a large dataset proposed for evaluation in the same work. At the same time, Khalifa and Shaalan (2019) used character Convolutional Neural Networks (CNN) as a replacement for character-level bidirectional Long Short-Term Memory (LSTM) in Arabic NER. Their proposed system was able to outperform the state-of-art systems, including character-level Bi-LSTM on various standard Arabic NER corpora. Antoun et al. (2020) proposed AraBERTv0.1 which involves pretraining the BERT transformer model for the Arabic language. They compared AraBERTv0.1 and the Bi-LSTM-CRF model on *ANERCorp*, the former achieved 84.2 F1 scores whereas the later achieved 81.7. Most recently, Sheng et al. (2020) proposed a transfer learning approach for Arabic NER with Deep Neural Networks where they showed that their model outperformed significantly the Bi-LSTM-CRF model. We have not considered this approach here because our aim is not to create a new state of the art model but to show the effectiveness of incorporating language specific characteristics in the form of embeddings.

Turning our attention now to Arabic POS tagging, many approaches have also been adopted over the years including rule-based methods (Alqrainy, 2008; Zribi et al., 2016), statistical models (Al Shamsi and Guessoum, 2006; Kadim and Lazrek, 2018), hybrid models (Vashishtha and Susan, 2019; Forsati and Shamsfard, 2014) and neural networks (Yousif and Sembok, 2006; Yousif and Sembok, 2005). Performance is usually much higher for POS tagging than NER. Khoja (2001) introduced a hybrid POS tagger (with 33 tags) which combined HMM with a rule-based tagger. They used the Holy Quran Corpus and achieved an accuracy rate of 97.6% and 96.8% respectively. Yousif and Sembok (2008) used the SVM approach and a corpus of 177 tagged words. Zeroual and Abdelhak (2016) presented a probabilistic POS tagger for Arabic text based on HMM called Tree Tagger. The proposed tagger obtained accuracy rates of 99.4% using Al-Mus'haf corpus.

Similar to other NLP applications, the most recent attention in this area has been on neural approaches. Wang et al. (2015) demonstrated an effective way of applying a Bi-LSTM to the POS tagging task, achieving 97.4% on the English Penn Treebank. Darwish et al. (2017) used a Bi-LSTM in their work on Arabic POS tagging, achieving 95.50%. Alrajhi and ELAffendi (2019) used the LSTM-RNN model on the Quranic Arabic Corpus (QAC). They reported accuracy of 99.76% at the word level and 99.18% at the morpheme level. They also compared their system against the Word2Vec POS tagger, for which they reported accuracy levels of 97.33% and 99.55% for words and morphemes respectively.

Returning to the different approaches of handling Arabic text. As discussed in the previous sections letter normalization and transliteration are examples of the simplification approach. For example, letter normalization is commonly applied to reduce the noise and sparsity in the data (Habash, 2010). For transliteration, Ameur et al. (2017) applied a bidirectional attention-based encoder-decoder model for the task of machine transliteration between Arabic and English.

Since the removal of diacritics also clearly leads to a potential ambiguity as explained in Section (2.1) there has been some work on automatic diacritization of partially diacritized or undiacritized text (Mubarak et al., 2019a; Mubarak et al., 2019b; Abdelali et al., 2016).

Shakkala was built by Barqawi (2017) for Arabic text diacritization using Bi-LSTM networks combined with character embeddings. Fadel et al. (2019) demonstrated the superiority of the neural approach of Shakkala compared to other different automatic diacritization systems available online e.g., Ali-Soft, Farasa, Harakat, and MADAMIRA.

Some recent work in Arabic NLP has started to make use of such systems. For example, Al-Sallab et al. (2017) proposed AROMA, a recursive deep learning model for opinion mining in Arabic. Preprocessing in AROMA included morphological tokenization and automatic diacritization carried out by

MADAMIRA (Pasha et al., 2014). This resulted in improved performance in classifying opinion as positive or negative on a range of different Arabic corpora. Similarly, Baly et al. (2017) used a Recursive Neural Tensor Network (RNTN) for sentiment analysis and reported that adding orthographic features such as diacritics improved the performance. They incorporated orthographic features such as diacritics by enlarging the vocabulary to have distinct word forms for different versions of the word (diacritized/undiacritized) and then deriving embeddings by training a Continuous Bag of Words (CBOW) model (Mikolov et al., 2013). Similarly, Alqahtani et al. (2019) introduced automatic selective diacritization as a viable step in lexical disambiguation. They evaluated the system in downstream tasks including POS which improved from 97.99% by baseline to 98.70%. They trained word embeddings on selectively-diacritized dataset to enrich the vocabulary.

3 Hypothesis

The hypothesis of this research can be summarized thus:

1. Incorporating linguistic characteristics of Arabic text in the form of embeddings can be exploited by a neural network, thus improving performance in downstream tasks.

2. Inferring linguistic characteristics of Arabic text to use as embedded features in a downstream model can improve downstream performance.

We have already outlined in Section (1) that there are two approaches of handling Arabic text, the simplification approach which involves transforming the text into simplified representation and the enrichment approach which minimally modifies the text and potentially adds more exploitable features to the text. We have assumed that the simplification approach might exclude some useful information and hence have adopted an enrichment approach. More specifically, we have adopted a pipeline approach where we use one model to infer a particular linguistic characteristic and then use the inferred information as features further downstream. For example, we infer diacritic information, using existing models such as Shakkala. We then exploit these features of the text in the form of embeddings in order to enhance performance in POS-tagging. We then use diacritics inferred by Shakkala and POS information inferred by POS model to enhance performance in NER.

We reason that using minimally modified Arabic text as input to neural networks builds the potential for allowing the neural network to learn an enriched representation of the Arabic language. Further, a framework such as EMIL, which incorporates more language-specific characteristics of the text in the form of embeddings should result in improved performance. In particular, since Arabic syntax and word sense disambiguation relies heavily on diacritics, we reason that applying an automatic diacritization neural model to minimally transformed Arabic text can capture syntactic and semantic dependencies. Similarly, useful information for NER can also be derived via POS tagging. Finally, we hypothesize that incorporating embedding layers based on derived language characteristics such as diacritics and POS tags can improve the overall performance of a neural network in sequence labeling tasks.

4 Approach

We propose a three-step approach to Arabic sequence labeling. The first step is to automatically diacritize the text using the state-of-the-art automatic diacritization system Shakkala (Barqawi, 2017). The second step is the individual training of character and diacritic embeddings using the architecture proposed by Gridach (2016). The third step is to train all embedding layers together using a combination model (see section 4.3). There are two main advantages in adopting this architecture for EMIL. First, it is based on a standard approach in NER and sequence labeling in general, which remains very close to the state-of-the-art. Second, it is a relatively light-weight architecture requiring less computational resources than other alternatives (see section 5.3). We will discuss and justify our design choices and the computational aspects of the architecture further in Section (5.3) and Section (6). Figure (2) gives an overview of the overall training procedure for our EMIL framework, which we now explain in detail.

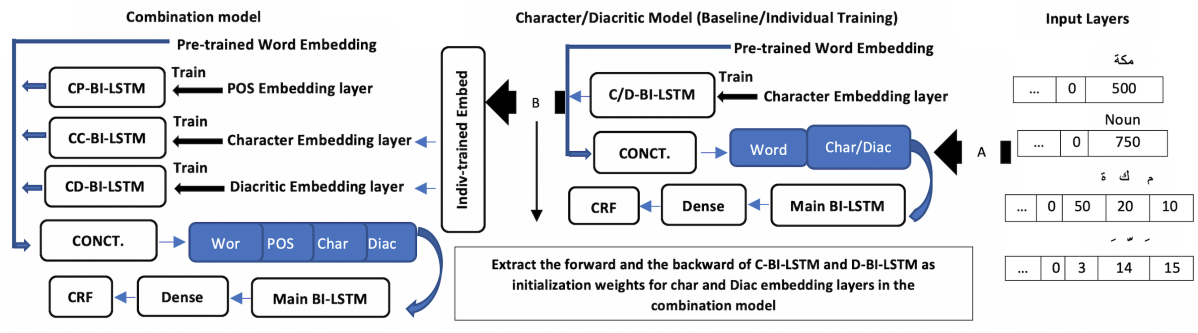

Figure 2: EMIL Training procedure.

4.1 Data Preparation

As shown in Figure (2), the *input layers* of our system are prepared with up to four types of input: word, POS, character and diacritic. The subset of input layers used depends on the task and setting. For example, when individually train the character model, we use the word and character input layers whereas when individually train the diacritic model, we use the word and diacritic input layers. For the final combination model for NER, we use all four input layers.

Since our data consists of variable length sentences, we use padding of 50 for the word and POS layers and 10 for the character and diacritic layers, which is consistent with the literature in this area. Thus, for each of the word and POS layers, the input has a shape of (50,). For each of the character and diacritic layers, the input has the shape (50,10) dimensions, which means for each word in the sentence there will be a 10 dimensional array representing the characters or diacritics respectively.

4.2 Individual-training: Character and Diacritic Models

Gridach (2016) proposed a character aware neural network model using a CRF on top of a Bi-LSTM. The aim of that model was to predict the NER tags by exploiting word and character embeddings. In our approach, we follow the same architecture. First, as shown in Figure (2) we use the *character model* directly to individually train character embeddings. The inputs to this model are word and character input layers as indicated by arrow (A). The word embedding layer takes as input pre-trained embedding matrix developed by Soliman et al. (2017), transforming the word input layer into word embeddings. The character embedding layer is randomly initialized and trained by the `C-Bi-LSTM`. The forward and the backward output from this `C-Bi-LSTM` is concatenated with the output from the word embedding layer and passed to the `main Bi-LSTM`. The output from this is passed to a `Dense` layer which maps the output of the `main Bi-LSTM` to the `CRF` layer, following Lample et al. (2016). After training the character model, we extract the forward and the backward output of the `C-Bi-LSTM` and use them to initialize the character embedding layer in the combination model.

The same approach is applied on *diacritic model* instead of using character information we use diacritics and instead of using `C-Bi-LSTM` we use `D-Bi-LSTM`. We extract the forward and backward outputs of the trained `D-Bi-LSTM` as individually-trained diacritic embeddings. It is worth noting that both of these models are trained on diacritized version of the datasets. Also it is important to mention that the output from this step are weights to initialize the character and diacritic embedding layers in the combination model and both of these sets of weights have been trained individually in separate models. The justification of this step can be found in Section (6) Table (2) where the experiments showed that training these embeddings separately and using them as individually-trained embedding in the final model improves the performance.

4.3 Combination model

As also shown in Figure (2), the final step in the EMIL training procedure is to combine all embedding layers and train them all together in a *combination model*. The first version of this model uses three input

layers (word, character and diacritic) and is referred to as the *character-diacritic model*. The second version uses all four layers of input and is referred to as the *four-layer combination model*. We now discuss the embeddings corresponding to each input layer in turn. First, the word embedding layer is provided with pre-trained word embeddings. Second, the optional POS embedding layer is randomly initialized (as shown in Figure (2), there is no arrow coming from the individually-trained embedding) and trained using `CP-Bi-LSTM`. Third, the character embedding layer is initialized with the individually-trained character embedding and re-trained using `CC-Bi-LSTM`. Fourth, the diacritic embedding layer is initialized with individually-trained diacritic embedding and re-trained using `CD-Bi-LSTM`. The output from the pre-trained word embedding, POS embedding, and the forward and backward output from CC-Bi-LSTM and CD-Bi-LSTM layers are concatenated by stacking each level on top of the other. The `main Bi-LSTM` layer is then trained using the concatenated layer, and the output of that layer is passed to the `dense` layer and from there to the `CRF` layer.

5 Empirical Evaluation

The *character model* described previously is used both as a baseline and for training the individually-trained character embedding. We also empirically evaluate two versions of the *combination model*: the *character-diacritic model* which incorporates word, character, and diacritic embedding layers; and the *four-layer combination model* which has an additional POS layer. We evaluate our models on two sequence labeling tasks: NER and POS tagging. We now describe the datasets used Section (5.1), hyperparameter settings Section (5.2) and the experiments performed to test our hypotheses Section (5.3). We used k-fold cross validation in order to evaluate the statistical significance of our results. The value of k used in k-fold cross validation varied according to the size of the dataset but was between 4 and 10 in all cases.

5.1 Datasets

We evaluate on 7 sequence labelling benchmarks: 4 of which are NER datasets and 3 of which are POS-tagging datasets[1]. In brief, each dataset consists of a sequence of sentences, where each sentence has a sequence of (word:tag) pairs. For NER we use the *BinAjeeba* (Darwish, 2013), the *ANERCorp* developed by Benajiba et al. (2007), and the *Wikipedia* and *Newswire* datasets which are mapped[2] versions of the fine-grained WikiFANE and NewsFANE datasets (Alotaibi and Lee, 2014) respectively. For POS-tagging we are evaluating on 3 standard datasets: *WikiNews* (Abdelali et al., 2019), *Al-Mushaf* (Zeroual and Abdelhak, 2016), *Prague Arabic Dependency Tree Bank (PADT)* (Hajic et al., 2004).

5.2 Hyper-parameters settings

Here, we are going to discuss the main hyper-parameters of the *combination model*. We did not tune these hyper-parameters — instead selecting values for computational efficiency (see section 5.3) and based on best practice from the literature. The `CP-Bi-LSTM`, `CC-Bi-LSTM` and `CD-Bi-LSTM` have 10 units each and the `main Bi-LSTM` has 20 units. Before we feed the output to the `CRF` layer, we use a `dense` layer with 20 units and tanh activation function to map the output of the `main Bi-LSTM` to the `CRF` layer. We use Keras which is an open source python library based on tensorflow. For optimization we used adam optimization technique [3].

Regarding the embeddings themselves, we used pre-trained word embeddings developed by Soliman et al. (2017) with 100 dimensions. The POS, character and diacritic embeddings have 10 dimensions each. The reason behind choosing 10 as the size of these embedding is that the size of the associated type vocabulary for each is small. For example, the Arabic alphabet contains 28 characters. We argue that 10 dimensional embedding will be enough to encode the information provided by the 28 character Arabic alphabet[4]. The word embedding is pre-trained, character and diacritic embeddings are individually-trained whereas the POS layer is randomly initialized. We set all of these embeddings to be

[1] For more details about the data sets, see Appendix A

[2] For the mapping, see Appendix A.1

[3] For more details about parameter settings, see Appendix B

[4] We also experimented with 25 dimensional embeddings as used by Gridach (2016) and found no further improvement

Corpus	Character model			Character-Diacritic model			Four-layer Combination model		
	F1	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision
ANERCorp (NER)	0.708	0.762	0.662	0.818*	0.833	0.803	**0.929****	0.938	0.920
BinAjeeba (NER)	0.720	0.763	0.681	0.840*	0.854	0.826	**0.930****	0.936	0.924
NewsWire (NER)	0.612	0.718	0.534	**0.751***	0.803	0.706	-	-	-
Wikipedia (NER)	0.758	0.794	0.726	**0.834***	0.851	0.819	-	-	-
Al Mushaf (POS)	0.941	0.940	0.942	**0.980***	0.9808	0.9806	-	-	-
PADT (POS)	0.94	0.941	0.939	**0.954***	0.955	0.953	-	-	-
WikiNews (POS)	0.949	0.951	0.948	**0.958***	0.960	0.956	-	-	-

Table 1: Summary of performance by each model on each dataset in the tasks. Best results are highlighted in bold. * indicates statistical significance at the 5% level in a paired sample comparison with the character model; ** with the character-diacritic model

trainable so they are trained together. We experimented with variations (randomly initialised character and diacritic embeddings (no individual-training), randomly initialised word embeddings, individually-trained POS embeddings and frozen embedding layers) during our ablation studies Section (6) and found this configuration of embeddings to be optimal.

5.3 Experiment

We conducted our experiments in five steps. First, as a baseline, we constructed a character aware model similar to the one in Figure (2) (*character model/baseline*) and trained it on the undiacritized version of the 7 datasets. It is worth noting that the dataset used to evaluate the baseline and the one used to evaluate the combination model are identical — the only difference between them is that the latter has been automatically diacritized using the Shakkala model.

Second, we used the diacritized version of the 7 datasets to extract the individually-trained character and diacritic embedding as described in Section (4.2) and passed it to the combination model. Third, we used the same diacritized version of the 7 datasets to train the *character-diacritic model* where we combine word, character and diacritic layers. The output from this sub-step is a diacritic-aware NER tagger for each of the 4 NER datasets and a diacritic-aware POS tagger for each of the 3 POS datasets. Fourth, we used our enhanced diacritic-aware POS tagger to tag 2 standard NER dataset (*BinAjeeba* and *ANERCorp*). Finally, we used these datasets to evaluate the *four-layer combination model* where we combine all four layers. It is worth noting that the amount of time required to cross-validate our models varied between 21 minutes and 387 minutes and the number of epochs required varied between 90 epochs and 340 epochs across all datasets. For example, cross-validating *ANERCorp* across the three models explained earlier took 77 minutes and 169 epochs for the *character model*, 85 minutes and 164 epochs for the *character-diacritic model* and 88 minutes and 240 epochs for the *four-layer combination model* which makes our architecture light-weight compared to BERT or other architectures which employ attention mechanisms[5].

6 Results

Table 1 summarises our results. For each model and dataset, we give the average precision, recall and F1-score calculated over the k folds of cross-validation. The *character model* is our baseline and is evaluated on the undiacritized version of the 7 datasets. We also give results for the *character-diacritic model* evaluated on the diacriticized versions of all 7 datasets and results for the *four-layer combination model* evaluated on automatically diacritized and automatically POS-tagged versions of 2 datasets.

First, we compare results for the *character model* and the *character-diacritic model* to see the impact of adding diacritics. We observe that the proposed *character-diacritic model* outperforms the *character model* on all of the datasets for both NER and POS-tagging. Automatically diacritizing the text and adding the diacritic embedding layer boosts both precision and recall of the system. We used k-fold cross-validation and all of the differences observed are statistically significant at the 5% level using a

[5]For more details about the computational aspects, see Appendix C

paired sample test. Our best absolute performance in NER was on the *BinAjeeba* dataset where we achieved an F1-score of 0.84, substantially outperforming the *character model* which achieved 0.72. Our best absolute and relative gain in performance on POS-tagging was on the *Al Mushaf* dataset where F1 performance rose from 0.941 to 0.980.

Second, we compare the *character-diacritic model* with the *four-layer combination model* to evaluate the impact of adding POS information as well as diacritics. We note that automatically POS tagging the diacritized NER datasets and adding this information via a POS embedding layer leads to a substantial gain in performance on both datasets. The performance on the *BinAjeeba* dataset improved form 0.84 to 0.93 using the *four-layer combination model*. Similarly, F1-score performance on the *ANERCorp* dataset rose from 0.82 to 0.92.

We also performed a number of ablation experiments on one of the benchmark datasets (*ANERCorp*) which are summarised in Table (2). It is worth noting that all the embedding layers in the ablation experiments are trainable apart from A5. In our ablations, we consider the effect of randomly initialising or training the different embedding layers for the combination model as follows: (A1) all embeddings randomly initialised, this ablation shows the importance of the pre-trained and the individual-trained components of the combination model. As shown in the table the performance drops substantially when we remove these components; (A2) word embeddings and POS randomly initialised, individually-trained character and diacritic embeddings, this ablation shows the significance of the pre-trained word embeddings and evidently proves its impact on the model. When we use randomly initialized word embeddings the performance drop from 0.929 to 0.611; (A3) pre-trained word embeddings, randomly initialized character, diacritics and POS embeddings, this ablation shows the impact of removing the individual training step from the combination model. The result shows that without the individual training of character and diacritic embedding the model achieves 0.747 however, when initializing character and diacritic embeddings of the combination model with individually-trained weights which are trained separately the performance is boosted to 0.929; and (A4) pre-trained word embeddings and the rest are individually trained, this ablation is to test the effectiveness of individually training POS embeddings and the results shows that this approach still outperformed by the original approach where we use randomly initialized POS embedding. We also considered (A5) freezing the embeddings rather than allowing them to be trainable in the final model but our results show the massive benefit of trainable embeddings. The previous ablation experiments supports the choice of our architecture where we use pre-trained word embedding along with randomly initialized pos embedding and individually trained character and diacritic embeddings.

EMIL	**A1**	**A2**	**A3**	**A4**	**A5**
0.929	0.617*	0.611*	0.747*	0.927	0.591*

Table 2: Summary of ablation results (F1-score); * indicates statistical significance at the 5% level in a paired sample comparison with EMIL

7 Error Analysis and Discussion

We now present a detailed analysis of the errors committed by each model on the *ANERCorp* dataset for the NER task. In our analysis, we identified 4 types of errors. **Type-one:** Boundary errors e.g., the word ديترويت (*Detroit*) appears once in the dataset tagged as B-ORG and is classified as I-ORG. **Type-two:** Low frequency words e.g., the word إسحق (*Isaac*) appears twice in the dataset tagged as B-PERS and is classified as O. We note that a lot of words in this type of error are foreign names. **Type-three:** Dominant tags e.g., the word السفير (*the ambassador*) is tagged twice as B-ORG and 12 times as O, and when this word appears as B-ORG a model classifies it as O. **Type-four:** Counter dominant tags e.g., the word مشهد (*Mashhad*) is labeled 17 times as B-LOC and once as O, and when it appears as B-LOC a model misclassifies it as O. We note that a lot of type-four errors appear to be gold standard errors where the word is tagged incorrectly and classified correctly by the model. For example, the word الضفة (*AL*

Daffa (The bank, like in riverbank)) is incorrectly tagged 4 times as O in the gold standard and our model classified it as B-LOC, which is in fact the correct label.

Model	Type 1	Type 2	Type 3	Type 4
(Baseline)	1540	2406	701	635
(Character-Diacritic)	889	1249	558	443
(Four-layer Combination)	214	683	117	60

Table 3: Summary of the errors committed by each model.

Table (3) shows a summary of the errors. We observe that the addition of diacritics reduced the number of errors in each type. Further, each error type was reduced again by the addition of POS information.

However, the largest reductions were for type-three and type-four errors which are both due to tag ambiguity. For example, the word السعودية (*Al Saudia*) can be tagged either as I-LOC or B-LOC depending on the context. Adding diacritics to this word will help disambiguate its sense. For example instances which should be tagged as B-LOC are adverbs and will be diacritized as السُّعُودِيَّةَ (*Al Saudiate*) whereas instances which should be tagged as I-LOC are adjectives and will be diacritized as السُّعُودِيَّةُ (*Al Saudiato*). It is worth noting that both words have the same meaning but each one plays a different syntactic and semantic role which can be encoded by the diacritics. This can be used as evidence to support our claim in Section (3) that neural network can learn syntactic and semantic information from the embeddings. In a perfect world, automatic diacritization will be optimal but we also note that Shakkala has some limitations. For example, foreign names written in Arabic, such as جينيرال موتور (*General Motors*), tend to produce errors, which is probably due to the sparsity of these types of names in Arabic corpora. Further, any errors committed by the automatic diacritization module tend to propagate through the sentence. Less frequently occurring diacritics such as gemination and nunation may also be misplaced. In some cases, the POS layer improved the performance as it was able to mitigate the errors produced by automatic diacritization. For example, if the word *Al Saudia* was incorrectly diacritized in the text, the POS tagger could mitigate this error by tagging it correctly as Adj or Adv based on its context. Of course, errors produced by the POS tagger will similarly affect the performance of the four-layer combination model As discussed in Section (3).

8 Conclusions and Future Work

In summary, we have shown that inferring and incorporating linguistic features of Arabic text in the form of embeddings can improve the performance of downstream tasks. Further, the pipeline approach that we are using may not be ideal regarding the quality of the inferred information but it still produces excellent results overall.

Focussing on diacritic information, we have shown that it is possible for a neural network to learn from the information encoded in the diacritics of the Arabic text and for this information to be used successfully in POS tagging and NER. By automatically diacriticising text and adding a diacritic-aware layer to existing neural architecture, performance (F1) was increased. We have also shown that adding POS information on top of diacritics in a similar way further improves performance at NER.

There are a number of directions for further work. First, further work on automatic diacritization, potentially directly including the diacritic inference model into the NER/POS training mode to reduce the effect of error propagation cause by the pipeline. Second, there is work to be done investigating the impact of embedding diacritic and POS information in attention-based architectures (Ali et al., 2019; Khalifa and Shaalan, 2019; Devlin et al., 2018).

Second, we believe that our approach of diacritic sensitive tagging will be useful in other areas of analysis including segmentation and deeper syntactic analysis. Further, embedding information from other analyses, including grammatical dependencies, might further improve performance in downstream tasks such as NER. Consequently, this is clearly only the beginning of investigating how the EMIL approach, where we *Embed More and Ignore Less*, can be applied to different sources of information, in different languages and to different downstream tasks.

References

Ahmed Abdelali, Kareem Darwish, Nadir Durrani, and Hamdy Mubarak. 2016. Farasa: A fast and furious segmenter for arabic. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: Demonstrations*, pages 11–16.

Ahmed Abdelali, Ali Elkahky, Hamdy Mubarak, Kareem Darwish, Mohammed Attia, and Younes Samih. 2019. Pos tagging for improving code-switching identification in arabic.

Samir AbdelRahman, Mohamed Elarnaoty, Marwa Magdy, and Aly Fahmy. 2010. Integrated machine learning techniques for arabic named entity recognition. *IJCSI*, 7:27–36.

Ahmed Abdul-Hamid and Kareem Darwish. 2010. Simplified feature set for arabic named entity recognition. In *Proceedings of the 2010 Named Entities Workshop*, pages 110–115. Association for Computational Linguistics.

Rabab Ali Abumalloh, Hassan Maudi Al-Sarhan, Othman Ibrahim, and Waheeb Abu-Ulbeh. 2016. Arabic part-of-speech tagging. *Journal of Soft Computing and Decision Support Systems*, 3(2):45–52.

Ahmad Al-Sallab, Ramy Baly, Hazem Hajj, Khaled Bashir Shaban, Wassim El-Hajj, and Gilbert Badaro. 2017. Aroma: A recursive deep learning model for opinion mining in arabic as a low resource language. *ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)*, 16(4):1–20.

Fatma Al Shamsi and Ahmed Guessoum. 2006. A hidden markov model-based pos tagger for arabic. In *Proceeding of the 8th International Conference on the Statistical Analysis of Textual Data, France*, pages 31–42.

Mohammed Nadher Abdo Ali, Guanzheng Tan, and Aamir Hussain. 2019. Boosting arabic named-entity recognition with multi-attention layer. *IEEE Access*, 7:46575–46582.

Fahd Alotaibi and Mark Lee. 2014. A hybrid approach to features representation for fine-grained arabic named entity recognition. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 984–995.

Sawsan Alqahtani, Hanan Aldarmaki, and Mona Diab. 2019. Homograph disambiguation through selective diacritic restoration. In *Proceedings of the Fourth Arabic Natural Language Processing Workshop*, pages 49–59, Florence, Italy, August. Association for Computational Linguistics.

Shihadeh Alqrainy. 2008. A morphological-syntactical analysis approach for arabic textual tagging.

Khwlah Alrajhi and Mohammed A ELAffendi. 2019. Automatic arabic part-of-speech tagging: Deep learning neural lstm versus word2vec. *International Journal of Computing and Digital Systems*, 8(03):307–315.

Mohamed Seghir Hadj Ameur, Farid Meziane, and Ahmed Guessoum. 2017. Arabic machine transliteration using an attention-based encoder-decoder model. *Procedia Computer Science*, 117:287–297.

Wissam Antoun, Fady Baly, and Hazem Hajj. 2020. Arabert: Transformer-based model for arabic language understanding. *arXiv preprint arXiv:2003.00104*.

Ramy Baly, Hazem Hajj, Nizar Habash, Khaled Bashir Shaban, and Wassim El-Hajj. 2017. A sentiment treebank and morphologically enriched recursive deep models for effective sentiment analysis in arabic. *ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)*, 16(4):1–21.

Zerrouki Barqawi. 2017. Shakkala, arabic text vocalization.

Yassine Benajiba and Paolo Rosso. 2007. Anersys 2.0: Conquering the ner task for the arabic language by combining the maximum entropy with pos-tag information. In *IICAI*.

Yassine Benajiba, Paolo Rosso, and José Miguel Benedíruiz. 2007. Anersys: An arabic named entity recognition system based on maximum entropy. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 143–153. Springer.

Yassine Benajiba, Mona Diab, Paolo Rosso, et al. 2008a. Arabic named entity recognition: An svm-based approach. In *Proceedings of 2008 Arab International Conference on Information Technology (ACIT)*, pages 16–18. Association of Arab Universities Amman, Jordan.

Yassine Benajiba, Mona Diab, Paolo Rosso, et al. 2008b. Arabic named entity recognition: An svm-based approach. In *Proceedings of 2008 Arab International Conference on Information Technology (ACIT)*, pages 16–18. Association of Arab Universities Amman, Jordan.

Kareem Darwish, Hamdy Mubarak, Ahmed Abdelali, and Mohamed Eldesouki. 2017. Arabic pos tagging: Don't abandon feature engineering just yet. In *Proceedings of the Third Arabic Natural Language Processing Workshop*, pages 130–137.

Kareem Darwish. 2013. Named entity recognition using cross-lingual resources: Arabic as an example. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1558–1567.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ali Fadel, Ibraheem Tuffaha, Bara' Al-Jawarneh, and Mahmoud Al-Ayyoub. 2019. Arabic text diacritization using deep neural networks. *arXiv preprint arXiv:1905.01965*.

Ali Farghaly and Khaled Shaalan. 2009. Arabic natural language processing: Challenges and solutions. *ACM Transactions on Asian Language Information Processing (TALIP)*, 8(4):1–22.

Rana Forsati and Mehrnoush Shamsfard. 2014. Hybrid pos-tagging: A cooperation of evolutionary and statistical approaches. *Applied Mathematical Modelling*, 38(13):3193–3211.

Mourad Gridach. 2016. Character-aware neural networks for arabic named entity recognition for social media. In *Proceedings of the 6th workshop on South and Southeast Asian natural language processing (WSSANLP2016)*, pages 23–32.

Nizar Habash, Ryan Roth, Owen Rambow, Ramy Eskander, and Nadi Tomeh. 2013. Morphological analysis and disambiguation for dialectal arabic. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 426–432.

Nizar Y Habash. 2010. Introduction to arabic natural language processing. *Synthesis Lectures on Human Language Technologies*, 3(1):1–187.

Jan Hajic, Otakar Smrz, Petr Zemánek, Jan Šnaidauf, and Emanuel Beška. 2004. Prague arabic dependency treebank: Development in data and tools. In *Proc. of the NEMLAR Intern. Conf. on Arabic Language Resources and Tools*, pages 110–117.

Ayoub Kadim and Azzeddine Lazrek. 2018. Parallel hmm-based approach for arabic part of speech tagging. *Int. Arab J. Inf. Technol.*, 15(2):341–351.

Muhammad Khalifa and Khaled Shaalan. 2019. Character convolutions for arabic named entity recognition with long short-term memory networks. *Computer Speech & Language*, 58:335–346.

Shereen Khoja. 2001. Apt: Arabic part-of-speech tagger. In *Proceedings of the Student Workshop at NAACL*, pages 20–25.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

Slim Mesfar. 2007. Named entity recognition for arabic using syntactic grammars. In *International Conference on Application of Natural Language to Information Systems*, pages 305–316. Springer.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Hamdy Mubarak, Ahmed Abdelali, Kareem Darwish, Mohamed Eldesouki, Younes Samih, and Hassan Sajjad. 2019a. A system for diacritizing four varieties of arabic. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, pages 217–222.

Hamdy Mubarak, Ahmed Abdelali, Hassan Sajjad, Younes Samih, and Kareem Darwish. 2019b. Highly effective arabic diacritization using sequence to sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2390–2395.

David Nadeau and Satoshi Sekine. 2007. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26.

Arfath Pasha, Mohamed Al-Badrashiny, Mona Diab, Ahmed El Kholy, Ramy Eskander, Nizar Habash, Manoj Pooleery, Owen Rambow, and Ryan Roth. 2014. MADAMIRA: A fast, comprehensive tool for morphological analysis and disambiguation of Arabic. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 1094–1101, Reykjavik, Iceland, May. European Language Resources Association (ELRA).

Khaled Shaalan and Hafsa Raza. 2007. Person name entity recognition for arabic. In *Proceedings of the 2007 Workshop on Computational Approaches to Semitic Languages: Common Issues and Resources*, pages 17–24. Association for Computational Linguistics.

Khaled Shaalan, Sanjeera Siddiqui, Manar Alkhatib, and Azza Abdel Monem. 2019. Challenges in arabic natural language processing. *Computational Linguistics*.

Khaled Shaalan. 2014. A survey of arabic named entity recognition and classification. *Computational Linguistics*, 40(2):469–510.

Jiabao Sheng, Aishan Wumaier, and Zhe Li. 2020. Poise: Efficient cross-domain chinese named entity recognization via transfer learning. *Symmetry*, 12(10):1673.

Abu Bakr Soliman, Kareem Eissa, and Samhaa R El-Beltagy. 2017. Aravec: A set of arabic word embedding models for use in arabic nlp. *Procedia Computer Science*, 117:256–265.

Srishti Vashishtha and Seba Susan. 2019. Fuzzy rule based unsupervised sentiment analysis from social media posts. *Expert Systems with Applications*, 138:112834.

Peilu Wang, Yao Qian, Frank K Soong, Lei He, and Hai Zhao. 2015. Part-of-speech tagging with bidirectional long short-term memory recurrent neural network. *arXiv preprint arXiv:1510.06168*.

Vikas Yadav and Steven Bethard. 2018. A survey on recent advances in named entity recognition from deep learning models. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2145–2158, Santa Fe, New Mexico, USA, August. Association for Computational Linguistics.

Jabar H Yousif and T Sembok. 2005. Arabic part-of-speech tagger based neural networks. In *proceedings of International Arab Conference on Information Technology ACIT2005, ISSN*, volume 857.

Jabar H Yousif and T Sembok. 2006. Design and implement an automatic neural tagger based arabic language for nlp applications. *Asian Journal of Information Technology*, 5(7):784–789.

Jabar Hassan Yousif and Tengku Mohd Tengku Sembok. 2008. Arabic part-of-speech tagger based support vectors machines. In *2008 International Symposium on Information Technology*, volume 3, pages 1–7. IEEE.

Imad Zeroual and Lakhouaja Abdelhak. 2016. Adapting a decision tree based tagger for arabic. In *2016 International Conference on Information Technology for Organizations Development (IT4OD)*, pages 1–6. IEEE.

Inès Zribi, Inès Kammoun, Mariem Ellouze, L Belguith, and Philippe Blache. 2016. Sentence boundary detection for transcribed tunisian arabic. *Bochumer Linguistische Arbeitsberichte*, 323.

9 Appendix

A Dataset

In this section we are going to present some statistics about the 7 datasets used in this paper. The data in this section is divided based on the tags that the datasets contain. For example, BinAjeeba and Wikipedia are gathered together because they have the same tags, and similarly, ANERCorp and NewsWire are gathered together. For POS, each corpus has its own POS tags so each data will be discussed separately.

Corpus	B-LOC	I-LOC	B-PERS	O	I-ORG	B-ORG	I-PERS
BinAjeeba	3553	1647	2910	501	1119	2342	98660
Wikipedia	46232	8615	22715	14732	9962	29895	881913

Table 4: Summary statistics of BinAjeeba and Wikipedia.

Table (4) shows the summary statistics of BinAjeeba and Wikipedia datasets. The *BinAjeeba* dataset contains 3889 sentences and 110732 tokens written in Modern Standard Arabic (MSA). The *Wikipedia*

contains 31397 sentences, 1014064 tokens written in MSA and are the mapped versions of the fine-grained WikiFANE. We mapped the fine grained tags present in these datasets onto the 3 broad-grained tags similar to the ones in the table. Both of these datasets are following Automatic Content Extraction (ACE) tagging guidelines with three types of named entities: location, person and organisation.

Corpus	B-LOC	I-LOC	B-PERS	O	B-MISC	I-ORG	I-MISC	B-ORG	I-PERS
ANERCorp	4269	1070	1914	3440	586	468	1298	2702	117822
NewsWire	4631	906	2325	3521	1220	279	2127	2939	139537

Table 5: Summary statistics of ANERCorp and NewsWire.

Table (5) shows the summary statistics of ANERCorp and NewsWire datasets. The *ANERCorp* contains 3889 sentences and 133569 tokens written in MSA. The *Newswire* contains 4886 sentences, 157485 tokens and are mapped versions of the fine-grained NewsFANE datasets. We mapped the fine grained tags present in this dataset onto the 4 broad-grained tags similar to the ones in the table. Both of these datasets have three standard tags: person, location, organization and also a fourth miscellaneous tag and follows ACE tagging guidelines.

For POS-tagging we are evaluating on 3 standard datasets: *WikiNews* has 571 sentences, 29992 tokens and 27 POS tags see Table (6); *Al-Mushaf* has 6347 sentences, 84593 tokens and 9 POS tags see Table (7); *Prague Arabic Dependency Tree Bank (PADT)* has 7609 sentences, 282384 tokens and 16 POS tags see Table (8).

Tag	Number of instances
NSUFF	17
FOREIGN	2077
FUT_PART	1
NOUN	2
ADJ	5
PART	45
NSUFF/ADJ	339
CASE	1323
ADV	4317
PRON	45
CONJ	63
PART/CONJ	8840
NSUFF/DET	2
PREP	3367
NOUN/DET	16
DET	3
NUM	3
ABBREV	425
NSUFF/NOUN	974
V	48
ADJ/NUM	25
ADJ/CONJ	3
PART/PREP+PART	8
PUNC	2920
PART/PART	1512
ADJ/DET	1754
PART/NOUN	1858

Table 6: Summary statistics of WikiNews dataset.

PRT	VERB	PUNC	DSIL	ADV	PN	NOUN	PRON	ADJ
978	1730	30	39113	948	1380	15455	6346	18613

Table 7: Summary statistics of AlMushaf dataset.

Tag	Number of instances
VERB	29351
SYM	42555
INTJ	1071
CCONJ	2165
DET	25241
NUM	5896
AUX	8
PART	93705
ADV	7758
ADP	2190
PROPN	10877
X	245
PUNCT	22445
NOUN	388
PRON	21300
ADJ	17189

Table 8: Summary statistics of PADT dataset.

A.1 Wikipedia and NewsWire Mapping

As mentioned in the previous section, both Wikipedia and NewsWire are mapped versions of the fine-grained WikiFANE and NewsFANE datasets. Both of WikiFANE and NewsFANE are following the (inside-outside-beginning) IOB format. WikiFANE has 103 tag in IOB format and 53 distinct tag trained on Wikipedia text. NewsFANE has 88 tags in IOB format and 46 distinct tags trained on NewsWire. We manually mapped each dataset by looking at the original tag and what it represents in order to map it to the equivalent tag. For example, FAC_Airport tag represents a location in the original dataset, hence it is mapped to location in the mapped dataset. For WikiFANE, we mapped each tag to the equivalent tags of BinAjeeba tagset which uses: Person, Location and Organization NEs, for NewsFANE we mapped each tag to the equivalent tags of ANERCorp tagset which uses: Person, Location, Organization and Miscellaneous NEs to create balance between datasets. It is worth noting that both WikiFANE and NewsFANE share the same level of granularity with different naming conventions and some extra tags. Table (9) shows a subset of the mapping in both Wikipedia and NewsWire.

WikiFANE Tag	Wikipedia Map	NewsFANE Tag	NewsWire Map
FAC_Airport	LOC	Airport	LOC
FAC_Building-Grounds	LOC	Building-Grounds	LOC
FAC_Path	LOC	Continent	LOC
FAC_Subarea-Facility	LOC	County-or-District	LOC
ORG_Commercial	ORG	Commercial	ORG
ORG_Educational	ORG	Educational	ORG
ORG_Entertainment	ORG	Entertainment	ORG
ORG_Government	ORG	Government	ORG
PER_Artist	PERS	Artist	PERS
PER_Athlete	PERS	Athlete	PERS
PER_Businessperson	PERS	Businessperson	PERS
PER_Engineer	PERS	Engineer	PERS
PRO_Drug	O	Drug	MISC
PRO_Food	O	Food	MISC
PRO_Hardware	O	Hardware	MISC
PRO_Movie	O	Movie	MISC

Table 9: Subset of the mapping of WikiFANE to Wikipedia and NewsFANE to NewsWire.

B Hyper-parameters Settings

In this section, we discuss the hyper-parameter settings in more detail. We used Keras which is an open-source neural-network library written in Python in the implementation of these models. For optimization, we used the adam optimization technique with a batch size of 32 batch, early stopping criteria based on the validation accuracy and validation split of 0.2 for all models. The models in our research can be divided in three types: baseline, individual-training models and combination model. In the following

subsections we are going to discuss it in details.

B.1 Baseline

In this section, we discuss the main hyper-parameters of the *baseline model*. We used pre-trained word embeddings with 100 dimensions and we set this layer to be trainable. The character embedding has 10 dimensions and we also set this layer to be trainable. The `C-Bi-LSTM` has 10 units wrapped with the `time distributed` Keras layer and 0.6 recurrent dropout. We concatenate the embeddings using Keras concatenation layer which stack each embedding on top of the other. The concatenated embedding here is the word embedding and the forward and the backward output of the `C-Bi-LSTM`. We also apply spatial dropout of 0.6 between the concatenation layer and the `main Bi-LSTM`. The `main Bi-LSTM` has 100 units with recurrent sequence sat to true and recurrent dropout of 0.6. We also placed a Keras dropout between the `main Bi-LSTM` and the `dense` layer. Before we feed the output to the `CRF` layer, we use a `dense` layer with 100 units, wrapped with `time distributed` Keras layer and tanh activation function to map the output of the `main Bi-LSTM` to the `CRF` layer. The size of the `CRF` layer is equal to the number of distinct tags.

B.2 Individual-Training Models for training character embedding

In this section we discuss the *character model* and the *diacritic model* that are used to individual-train character and diacritic embedding layers. These two models are similar to the baseline and following exactly the same architecture. The only difference between these two models and the baseline is that after we train these models we extract the forward and the backward output of the `C-Bi-LSTM` and `D-Bi-LSTM` to use it as initialization weights for character and diacritic embedding respectively for the *combination model*.

B.3 Combination Model

Here, we discuss the hyper-parameters of the *combination model*. This model has two versions *character-diacritic model* and the *four-layer combination model*. They both share the same parameters except two cases they are different, so we will mention the parameters all together and highlight the distinct parameters as we proceed. We used pre-trained word embeddings similar to the one mentioned earlier with 100 dimensions. The POS, character and diacritic embeddings have 10 dimensions each. The word embeddings are pre-trained, character and diacritic embeddings are individually-trained whereas the POS layer is randomly initialized. We set all of these embeddings to be trainable so they are trained together. In the *character-diacritic model* the `CC-Bi-LSTM` and `CD-Bi-LSTM` have 10 units with recurrent dropout of 0.6 each. In the *four-layer combination model* `CP-Bi-LSTM`, `CC-Bi-LSTM` and `CD-Bi-LSTM` have 10 units with recurrent dropout of 0.5 each. The `CC-Bi-LSTM` and `CD-Bi-LSTM` layers are wrapped with the `time distributed` Keras layer and the `CP-Bi-LSTM` is not. We concatenate the embeddings using Keras concatenation layer which stack each embedding on top of the other. In the *character-diacritic model* the concatenated embedding is the word embeddings and the forward and the backward output of the `CC-Bi-LSTM` and the `CD-Bi-LSTM` layers. In the *four-layer combination model*, the concatenated embedding is the word, POS embeddings and the forward and the backward output of the `CC-Bi-LSTM` and the `CD-Bi-LSTM` layers. We also apply spatial dropout of 0.6 between the concatenation layer and the `main Bi-LSTM` similar to the previous models. The `main Bi-LSTM` has 20 units with recurrent sequence set to true. We also placed a Keras dropout between the `main Bi-LSTM` and the `dense` layer. Before we feed the output to the `CRF` layer, we use a `dense` layer with 100 units, wrapped with a `time distributed` Keras layer and tanh activation function to map the output of the `main Bi-LSTM` to the `CRF` layer. The size of the `CRF` layer is equal to the number of distinct tags.

C Computational Aspects

In this section, we present the computational time and the number of epochs that each model took to train across all datasets. It is worth noting that we didn't compute the time and number of epochs of the

individual-training models since it is a one time cost. Table (10) shows the time and number of epochs needed for each model on each dataset, where time is in minutes. This table shows that our proposed model is light weight which is an advantage over the other heavy weights models such as BERT or attention.

Corpus	**Character model**		**Character-Diacritic model**		**Four-layer Combination model**	
	Time	**Epochs**	**Time**	**Epochs**	**Time**	**Epochs**
ANERCorp	77	169	85	164	88	240
BinAjeeba	69	176	65	224	21	194
NewsWire	143	261	67	276	-	-
Wikipedia	350	90	387	104	-	-
AlMushaf	154	318	122	306	-	-
PADT	145	229	101	136	-	-
WikiSeg	20	282	16	340	-	-

Table 10: The time in minutes and the number of epochs taken to train each model for each dataset.